UNIT III

Dynamic Programming: General method, applications- Optimal binary search trees, 0/1 knapsack problem, All pairs shortest path problem, Traveling sales person problem, Reliability design.

DYNAMIC PROGRAMMING

Dynamic programming is a name, coined by Richard Bellman in 1955. Dynamic programming, as greedy method, is a powerful algorithm design technique that can be used when the solution to the problem may be viewed as the result of a sequence of decisions. In the greedy method we make irrevocable decisions one at a time, using a greedy criterion. However, in dynamic programming we examine the decision sequence to see whether an optimal decision sequence contains optimal decision subsequence.

When optimal decision sequences contain optimal decision subsequences, we can establish recurrence equations, called *dynamic-programming recurrence equations*, that enable us to solve the problem in an efficient way.

Dynamic programming is based on the principle of optimality (also coined by Bellman). The principle of optimality states that no matter whatever the initial state and initial decision are, the remaining decision sequence must constitute an optimal decision sequence with regard to the state resulting from the first decision. The principle implies that an optimal decision sequence is comprised of optimal decision subsequences. Since the principle of optimality may not hold for some formulations of some problems, it is necessary to verify that it does hold for the problem being solved. Dynamic programming cannot be applied when this principle does not hold.

The steps in a dynamic programming solution are:

- Verify that the principle of optimality holds
- Set up the dynamic-programming recurrence equations
- Solve the dynamic-programming recurrence equations for the value of the optimal solution.
- Perform a trace back step in which the solution itself is constructed.

ALL PAIRS SHORTEST PATHS

In the all pairs shortest path problem, we are to find a shortest path between every pair of vertices in a directed graph G. That is, for every pair of vertices (i, j), we are to find a shortest path from i to j as well as one from j to i. These two paths are the same when G is undirected.

When no edge has a negative length, the all-pairs shortest path problem may be solved by using Dijkstra’s greedy single source algorithm n times, once with each of the n vertices as the source vertex.

The all pairs shortest path problem is to determine a matrix A such that A (i, j) is the length of a shortest path from i to j. The matrix A can be obtained by solving n single-source problems using the algorithm shortest Paths. Since each application of this procedure requires O (n^2) time, the matrix A can be obtained in O (n^3) time.

The dynamic programming solution, called Floyd's algorithm, runs in O (n^3) time. Floyd's algorithm works even when the graph has negative length edges (provided there are no negative length cycles).

The shortest i to j path in G, i ≠ j originates at vertex i and goes through some intermediate vertices (possibly none) and terminates at vertex j. If k is an intermediate vertex on this shortest path, then the subpaths from i to k and from k to j must be shortest paths from i to k and k to j, respectively. Otherwise, the i to j path is not of minimum length. So, the principle of optimality holds. Let A^k (i, j) represent the length of a shortest path from i to j going through no vertex of index greater than k, we obtain:

Ak (i, j) = {min {$\underline{\text{min}}$ {A^{k-1} (i, k) + A^{k-1} (k, j)}, c (i, j)} 1<k<n

Algorithm All Paths (Cost, A, n)
// cost [1:n, 1:n] is the cost adjacency matrix of a graph which
// n vertices; A [I, j] is the cost of a shortest path from vertex
// i to vertex j. cost [i, i] = 0.0, for 1 ≤ i ≤ n.

```
{
        for i := 1 to n do
                for j:= 1 to n do
                        A [i, j] := cost [i, j]; // copy cost into A. for k := 1 to n do

                for i := 1 to n do
                        for j := 1 to n do
                                A [i, j] := min (A [i, j], A [i, k] + A [k, j]);
}
```

Complexity Analysis: A Dynamic programming algorithm based on this recurrence involves in calculating n+1 matrices, each of size n x n. Therefore, the algorithm has a complexity of O (n^3).

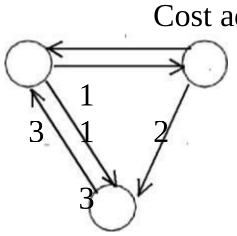
Example 1:

Given a weighted digraph G = (V, E) with weight. Determine the length of the shortest path between all pairs of vertices in G. Here we assume that there are no cycles with zero or negative cost.

6
1 2
4

0
Cost adjacency matrix (A) = ~6

r0 11
4 2
~ 0 ~
~
0
~
~L3 ~]



General formula: min {A^{k-1} (i, k) + A^{k-1} (k, j)}, c (i, j)} 1<k<n

Solve the problem for different values of k = 1, 2

and 3 **Step 1**: Solving the equation for, k = 1;
A1 (1, 1) = min {(A^o (1, 1) + A^o (1, 1)), c (1, 1)} = min {0 + 0, 0} = 0 A1 (1, 2) = min {(A^o (1, 1) + A^o (1, 2)), c (1, 2)} = min {(0 + 4), 4} = 4
A1 (1, 3) = min {(A^o (1, 1) + A^o (1, 3)), c (1, 3)} = min {(0 + 11), 11} = 11 A1 (2, 1) = min {(A^o (2, 1) + A^o (1, 1)), c (2, 1)} = min {(6 + 0), 6} = 6
A1 (2, 2) = min {(A^o (2, 1) + A^o (1, 2)), c (2, 2)} = min {(6 + 4), 0)} = 0 A1 (2, 3) = min {(A^o (2, 1) + A^o (1, 3)), c (2, 3)} = min {(6 + 11), 2} = 2 A1 (3, 1) = min {(A^o (3, 1) + A^o (1, 1)), c (3, 1)} = min {(3 + 0), 3} = 3 A1 (3, 2) = min {(A^o (3, 1) + A^o (1, 2)), c (3, 2)} = min {(3 + 4), oc} = 7 A1 (3, 3) = min {(A^o (3, 1) + A^o (1, 3)), c (3, 3)} = min {(3 + 11), 0} = 0

A(1)
= ~0 4 11
~ ~
~6 0 2 ~
~L 0
3 7 ~U

Step 2: Solving the equation for, K = 2;

A2 (1,

A2 (1,

A2 (1,

A2 (2,

A2 (2,

A2 (2,

A2 (3,

A2 (3,

A2 (3,

1) = min {(A^1 (1, 2) + A^1 (2, 1), c (1, 1)} = min {(4 + 6), 0} + A^1 = 0
2) = min {(A^1 (1, 2) (2, 2), c (1, 2)} = min {(4 + 0), 4} + A^1 (2, = 4 =
3) = min {(A^1 (1, 2) 3), c (1, 3)} = min {(4 + 2), 11} 6
1) = min {(A (2, 2) + A (2, 1), c (2, 1)} = min {(0 + 6), 6}= 6
2) = min {(A (2, 2) + A (2, 2), c (2, 2)} = min {(0 + 0), 0}= 0
3) = min {(A (2, 2) + A (2, 3), c (2, 3)} = min {(0 + 2), 2}= 2

1) = min {(A (3, 2) + A (2, 1), c (3, 1)} = min {(7 + 6), 3}= 3
2) = min {(A (3, 2) + A (2, 2), c (3, 2)} = min {(7 + 0), 7}= 7
3) = min {(A (3, 2) + A (2, 3), c (3, 3)} = min {(7 + 2), 0}= 0

A(2) = ~0 4 6 1
~ 2~
~6
3̃L 0 ~
0
~~
7

Step 3: Solving the equation for, k = 3;

A3 (1, 1) = min {A^2 (1, 3) + A^2 (3, 1), c (1, 1)} = min {(6 + 3), 0} = 0

A3 (1, 2) = min {A^2 (1, 3) + A^2 (3, 2), c (1, 2)} = min {(6 + 7), 4} = 4

A3 (1, 3) = min {A^2 (1, 3) + A^2 (3, 3), c (1, 3)} = min {(6 + 0), 6} = 6

A3 (2, 1) = min {A^2 (2, 3) + A^2 (3, 1), c (2, 1)} = min {(2 + 3), 6} = 5

A3 (2, 2) = min {A^2 (2, 3) + A^2 (3, 2), c (2, 2)} = min {(2 + 7), 0} = 0

A3 (2, 3) = min {A^2 (2, 3) + A^2 (3, 3), c (2, 3)} = min {(2 + 0), 2} = 2

A3 (3, 1) = min {A^2 (3, 3) + A^2 (3, 1), c (3, 1)} = min {(0 + 3), 3} = 3

A3 (3, 2) = min {A^2 (3, 3) + A^2 (3, 2), c (3, 2)} = min {(0 + 7), 7} = 7

107

A3 (3, 3) = min {A^2 (3, 3) + A^2 (3, 3), c (3, 3)} = min {(0 + 0), 0} = 0

A(3) = ~ 0 4 6 ~
~
0 2~
~~5~~3 7 0 ~]

TRAVELLING SALESPERSON PROBLEM:

Let G = (V, E) be a directed graph with edge costs Cij. The variable cij is defined such that cij > 0 for all I and j and cij = a if < i, j> o E. Let |V| = n and assume n > 1. A tour of G is a directed simple cycle that includes every vertex in V. The cost of a tour is the sum of the cost of the edges on the tour. The traveling sales person problem is to find a tour of minimum cost. The tour is to be a simple path that starts and ends at vertex 1.

Let g (i, S) be the length of shortest path starting at vertex i, going through all vertices in S, and terminating at vertex 1. The function g (1, V – {1}) is the length of an optimal
salesperson tour. From the principal of optimality it follows that:

g(1, V - {1 }) = 2 ~ *k* ~ *n* ~*c*1*k* ~ *g* ~ *k*, *V* ~ ~ 1, *k* ~~ -- 1
~

min
-- 2

Generalizing equation 1, we obtain (for i o S)

g (i, S) = min{*ci j*
j E*S*

The Equation can be solved for g (1, V – 1}) if we know g (k, V – {1, k}) for all choices of k.

Complexity Analysis:

For each value of |S| there + *g* (*i*, *S* - { *j* }) } are n – 1 choices for i. The number of distinct sets S of
size k not including 1 and i is I *k*~~ . ~ ~n -2 ~

Hence, the total number of g (i, S)'s to be computed before computing g (1, V – {1}) is:

~*n* -2 ~
~ ~1
~ ~ ~ ~
n ~ *k*

k ~ 0

~ ~

To calculate this sum, we use the binominal theorem:

[((*n* - 2) ((*n* - 2) ((*n* - 2) ((*n* - 2)1

n
-
1 (n – 1) 111 11+ii iI+ ii iI+ ----~ ~~ ~
~
~
0)~1)~ 2) ~(*n* ~2)~~

According to the binominal theorem:

[((*n* - 2) ((*n* - 2) ((*n* - 2 ((*n* - 2)1

1
1
+
i ~~~~~~~~~
il i iI+ii ~ ~~~ = 2n - 2
~~ ~(*n* -
0 ~~1~~2~ 2))]

Therefore,

n -
1 ~ *n* _ 2'
~ Ī
(*n* ~ ~
~ *k* = (*n* - 1) 2*n* ~ 2
~

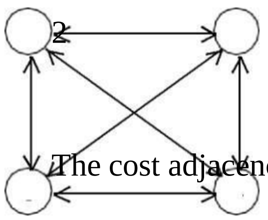
This is Φ (n 2^{n-2}), so there are exponential number of calculate. Calculating one g (i, S) require finding the minimum of at most n quantities. Therefore, the entire algorithm is Φ (n^2 2^{n-2}). This is better than enumerating all n! different tours to find the best one. So, we have traded on exponential growth for a much smaller exponential growth.

The most serious drawback of this dynamic programming solution is the space needed, which is O (n 2^n). This is too large even for modest values of n.

Example 1:

For the following graph find minimum cost tour for the traveling salesperson problem:

1 2



3 4

The cost adjacency matrix =

r
0 20
1 10
~ 0 15 ~
~ 0 9 ~
1 12
5 3 0 ~
~
6 01
8 9]
~

Let us start the tour from vertex 1:

g (1, V – {1}) = min {c1k + g (k, V – {1, K})} - (1)
2<k<n

More generally writing:

g (i, s) = min {cij + g (J, s – {J})} - (2)

Clearly, g (i, T) = ci1 , 1 ≤ i ≤ n. So,

g (2, T) = C21 = 5

g (3, T) = C31 = 6

g (4, ~) = C41 = 8

Using equation – (2) we obtain:

g (1, {2, 3, 4}) = min {c12 + g (2, {3, 4}, c13 + g (3, {2, 4}), c14 + g (4, {2, 3})}

g (2, {3, 4}) = min {c23 + g (3, {4}), c24 + g (4, {3})}

= min {9 + g (3, {4}), 10 + g (4, {3})}

g (3, {4}) = min {c34 + g (4, T)} = 12 + 8 = 20

g (4, {3}) = min {c43 + g (3, ~)} = 9 +6=15

Therefore, g (2, {3, 4}) = min {9 + 20, 10 + 15} = min {29, 25} = 25

g (3, {2, 4}) = min {(c32 + g (2, {4}), (c34 + g (4, {2})}

g (2, {4}) = min {c24 + g (4, T)} = 10 + 8 = 18

g (4, {2}) = min {c42 + g (2, ~)} = 8 + 5 = 13

Therefore, g (3, {2, 4}) = min {13 + 18, 12 + 13} = min {41, 25} = 25 g (4, {2, 3}) =

min {c42 + g (2, {3}), c43 + g (3, {2})}

g (2, {3}) = min {c23 + g (3, ~} = 9 + 6 = 15

g (3, {2}) = min {c32 + g (2, T} = 13+ 5 = 18

Therefore, g (4, {2, 3}) = min {8 + 15, 9 + 18} = min {23, 27} = 23

g (1, {2, 3, 4}) = min {c12 + g (2, {3, 4}), c13 + g (3, {2, 4}), c14 + g (4, {2, 3})} = min {10 + 25, 15 + 25, 20 + 23} = min {35, 40, 43} = 35

The optimal tour for the graph has length = 35 The

optimal tour is: 1, 2, 4, 3, 1.

OPTIMAL BINARY SEARCH TREE

Let us assume that the given set of identifiers is {a1, . . . , an} with a1 < a2 < . . . . < an. Let p (i) be the probability with which we search for ai. Let q (i) be the probability that the identifier x being searched for is such that ai < x < ai+1, 0 ≤ i ≤ n (assume a0 = -~ and an+1 = +oc). We have to arrange the identifiers in a binary search tree in a way that minimizes the expected total access time.

In a binary search tree, the number of comparisons needed to access an element at depth 'd'
is d + 1, so if 'ai' is placed at depth 'di', then we want to minimize:

$$\sum_{i=1}^{n} P_i (1 + d_i) .$$

Let P (i) be the probability with which we shall be searching for 'ai'. Let Q (i) be the probability of an un-successful search. Every internal node represents a point where a successful search may terminate. Every external node represents a point where an unsuccessful search may terminate.

The expected cost contribution for the internal node for 'ai' is:

$$P(i) * level(a_i) .$$

Unsuccessful search terminate with I = 0 (i.e at an external node). Hence the cost contribution for this node is:

$$Q(i) * level((E_i) - 1)$$

110

The expected cost of binary search tree is:

$$\sum P(i) * level(a_i) + \sum Q(i) * level((E_i) - 1)$$

Given a fixed set of identifiers, we wish to create a binary search tree organization. We may expect different binary search trees for the same identifier set to have different performance characteristics.

The computation of each of these c(i, j)'s requires us to find the minimum of m quantities. Hence, each such c(i, j) can be computed in time O(m). The total time for all c(i, j)'s with j – i = m is therefore O(nm – m^2).

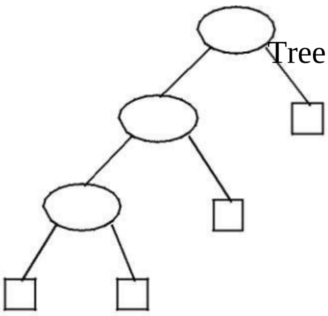
The total time to evaluate all the c(i, j)'s and r(i, j)'s is therefore:

$$\sum_{1 < m < n} (nm - m^2) = O(n^3)$$

Example 1: The possible binary search trees for the identifier set (a1, a2, a3) = (do, if, stop) are as follows. Given the equal probabilities p (i) = Q (i) = 1/7 for all i, we have:

st o p

if



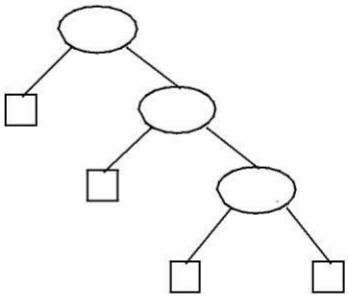
Tree 1

do

Tree 2

do

if
st
o
p



Tree 3

(1
(1 x 1 + 1 x 2 + 1 x 3~ x 1
Cost (tree # 1) = ~ + ~
~ 7
~ 7 7 7)

1+2+31+2+3+36+915

(1 x 1
Cost (tree # 3) = ~ x 1 + 1 x 2 + 1 x 3~ ~ + 1 x 2 x 3 + 1 x 3~
1 + ~ + 1 ~
~
~ 7 7 ~ 7
7) 7 7 7)
1+2+3 1+2+3+3 6+9 15
(
x~1 +
= 7 + 7 ~ 1 1
Cost (tree # 4) = ~ x 1 + 1 x 2 ~ 1 x 3~ ~ x 2 x 3 + 1 x 3~
1 ~ ~ + 1 ~
~ 7 ~7 7
~ 7) 7 7)
7
= 1 +2 +3 1 +2+3+3 6+9 15
7

7
(1 x 1 1 x
+ 1 2~ (1 x 2 + 1 x 2
x + 1 1 x 2~
2
Cost (tree # 2) = ~ + + ~ x 2
~
~ 7 7 7) 7 7 7 7)
5
2+2+2+ +
=1+2+2 + 2 ~ 8 ~ 13

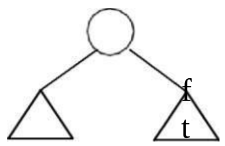
7

Huffman coding tree solved by a greedy algorithm has a limitation of having the data only at the leaves and it must not preserve the property that all nodes to the left of the root have keys, which are less etc. Construction of an optimal binary search tree is harder, because the data is not constrained to appear only at the leaves, and also because the tree must satisfy the binary search tree property and it must preserve the property that all nodes to the left of the root have keys, which are less.

A dynamic programming solution to the problem of obtaining an optimal binary search tree can be viewed by constructing a tree as a result of sequence of decisions by holding the principle of optimality. A possible approach to this is to make a decision as which of the ai's be arraigned to the root node at 'T'. If we choose 'ak' then is clear that the internal nodes for a1, a2, ...ak-1 as well as the external nodes for the classes Eo, E1,
. . . . Ek-1 will lie in the left sub tree, L, of the root. The remaining nodes will be in the right subtree, ft. The structure of an optimal binary search tree is:

ak

L



$$\text{Cost (L)} = \sum_{i=1}^{K} P(i) * level(a_i) + \sum_{i=0}^{K} Q(i) * (level(E_i) - 1)$$

$$\text{Cost (ft)} = \sum_{i=K}^{n} P(i) * level(a_i) + \sum_{i=K}^{n} Q(i) * (level(E_i) - 1)$$

The C (i, J) can be computed as:

C (i, J) = min {C (i, k-1) + C (k, J) + P (K) + w (i, K-1) + w (K, J)} i<k<J

= min {C (i, K-1) + C (K, J)} + w (i, J) -- (1)
i<k<J

Where W (i, J) = P (J) + Q (J) + w (i, J-1) -- (2)

Initially C (i, i) = 0 and w (i, i) = Q (i) for $0 \le i \le n$.

Equation (1) may be solved for C (0, n) by first computing all C (i, J) such that J - i = 1 Next, we can compute all C (i, J) such that J - i = 2, Then all C (i, J) with J - i = 3 and so on.
C (i, J) is the cost of the optimal binary search tree 'Tij' during computation we record the root R (i, J) of each tree 'Tij'. Then an optimal binary search tree may be constructed from these R (i, J). R (i, J) is the value of 'K' that minimizes equation (1).

We solve the problem by knowing W (i, i+1), C (i, i+1) and R (i, i+1), 0 ≤ i < 4;

Knowing W (i, i+2), C (i, i+2) and R (i, i+2), 0 ≤ i < 3 and repeating until W (0, n), C (0, n) and R (0, n) are obtained.

The results are tabulated to recover the actual tree.

Example 1:

Let n = 4, and (a1, a2, a3, a4) = (do, if, need, while) Let P (1: 4) = (3, 3, 1, 1) and Q (0: 4) = (2, 3, 1, 1, 1)

Solution:

Table for recording W (i, j), C (i, j) and R (i, j):

Column / Row	0	1	2	3	4
0	2,0,0	3,0, 0	1, 0, 0	1, 0, 0,	1,0,0
1	8,8,1	7,7, 2	3, 3, 3	3, 3, 4	
2	12, 19, 1	9, 12, 2	5, 8, 3		
3	14, 25, 2	11, 19, 2			
4	16, 32, 2				

This computation is carried out row-wise from row 0 to row 4. Initially, W (i, i) = Q (i) and C (i, i) = 0 and R (i, i) = 0, 0 ≤ i < 4. Solving for C (0, n):

First, computing all C (i, j) such that j - i = 1; j = i + 1 and as 0 ≤ i < 4; i = 0, 1, 2 and 3; i < k ≤ J. Start with i = 0; so j = 1; as i < k ≤ j, so the possible value for k = 1

W(0,1)=P(1)+Q(1)+W(0,0)=3+3+2=8
C (0, 1) = W (0, 1) + min {C (0, 0) + C (1, 1)} = 8
R (0, 1) = 1 (value of 'K' that is minimum in the above equation).

Next with i = 1; so j = 2; as i < k ≤ j, so the possible value for k = 2

W (1, 2) =P(2)+Q(2)+W(1, 1) =3+1 +3= 7
C 2 = W (1, 2) + min {C 1) 2)}

(1,) (1, +C(2, =7
R 2
(1,) = 2

Next with i = 2; so j = 3; as i < k ≤ j, so the possible value for k = 3

W (2, 3) =P(3)+Q(3)+W(2, 2) =1+1 +1= 3
C (2, 3) = W (2, 3) + min {C (2, 2) +C(3, 3)}=3 + [(0 + 0)] =3
ft (2, 3) = 3

Next with i = 3; so j = 4; as i < k ≤ j, so the possible value for k = 4
W (3, 4) =P(4)+Q(4)+W(3,3) =1+1 +1=3
C (3, 4) = W (3, 4) + min {[C (3, 3) + C (4, 4)]} =3 + [(0 + 0)] =3
ft (3, 4) = 4

Second, Computing all C (i, j) such that j - i = 2; j = i + 2 and as 0 ≤ i < 3; i = 0, 1, 2; i < k ≤ J. Start with i = 0; so j = 2; as i < k ≤ J, so the possible values for k = 1 and 2.

W(0,2)=P(2)+Q(2)+W(0,1)=3+1+8=12
C (0, 2) = W (0, 2) + min {(C (0, 0) + C (1, 2)), (C (0, 1) + C (2, 2))} = 12
+ min {(0 + 7, 8 + 0)} = 19 ft (0, 2) = 1
Next, with i = 1; so j = 3; as i < k ≤ j, so the possible value for k = 2 and 3.
W(1, 3) = P (3) +Q(3)+W(1,2)=1+1+7=9
C (1, 3)= W (1, 3) + min {[C (1, 1) + C (2, 3)], [C (1, 2) + C (3, 3)]}
=W(1, 3) + min {(0 + 3), (7 + 0)} = 9 + 3 = 12
ft (1, 3) = 2

Next, with i = 2; so j = 4; as i < k ≤ j, so the possible value for k = 3 and 4.

W(2,4)=P(4)+Q(4)+W(2,3)=1+1+3=5
C (2, 4) = W (2, 4) + min {[C (2, 2) + C (3, 4)], [C (2, 3) + C (4, 4)]
= 5 + min {(0 + 3), (3 + 0)} = 5 + 3 = 8 ft (2, 4) = 3

Third, Computing all C (i, j) such that J - i = 3; j = i + 3 and as 0 ≤ i < 2; i = 0, 1; i < k ≤ J. Start with i = 0; so j = 3; as i < k ≤ j, so the possible values for k = 1, 2 and 3.
W (0, 3) =P(3)+Q(3)+W(0,2)=1+1=+ 12 = 14
C (0, 3) W (0, 3) + min {[C (0, 0) + C (1, 3)], [C (0, 1) + C (2, 3)],

[C(0,2)+C(3,= 3)]}
+0)}=14 +11=25

ft
(0, 14 + min {(0 + 12), (8 + 3), (19 =
3) 2

Start with i = 1; so j = 4; as i < k ≤ j, so the possible values for k = 2, 3 and 4.

W (1, 4) =P(4)+Q(4)+W(1,3)=1+1+9=11=W 4)
C (1, 4) (1, 4) + min {[C (1, 1) + C (2, 4)], [C (1, 2) +C(3, 4)],
[C (1, 3) + C (4, 4)]}
+8= 19

ft
(1, = 11 + min {(0 + 8), (7 + 3), (12 + 0)} = 11 =
4) 2

Fourth, Computing all C (i, j) such that j - i = 4; j = i + 4 and as 0 ≤ i < 1; i = 0; i < k ≤ J.

Start with i = 0; so j = 4; as i < k ≤ j, so the possible values for k = 1, 2, 3 and 4.

W (0, 4) =P(4) +Q(4)+W(0, 3) =1+1 +14=16

C (0, 4) =W(0, 4) + min {[C (0, 0) +C(1, 4)], [C (0, 1) +C(2, 4)],
[C (0, 2) +C(3, 4)], [C (0, 3) +C(4, 4)]}
= 16 + min [0 + 19, 8 + 8, 19+3, 25+0] = 16 + 16 = 32 ft (0,
4)=2

From the table we see that C (0, 4) = 32 is the minimum cost of a binary search tree for (a1, a2, a3, a4). The root of the tree 'T04' is 'a2'.

Hence the left sub tree is 'T01' and right sub tree is T24. The root of 'T01' is 'a1' and the root of 'T24' is a3.

The left and right sub trees for 'T01' are 'T00' and 'T11' respectively. The root of T01 is 'a1'

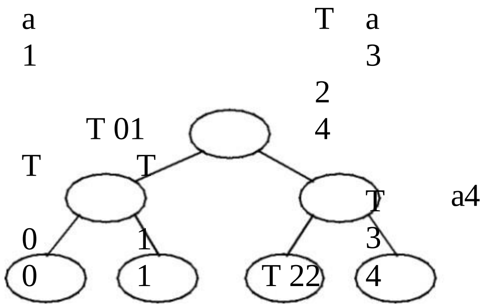
The left and right sub trees for T24 are T22 and T34 respectively.

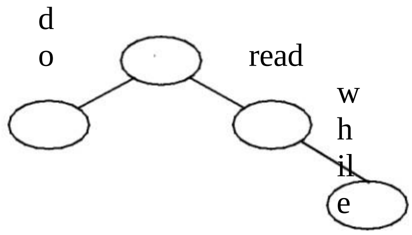
The root of T24 is 'a3'.

The root of T22 is null

The root of T34 is

a2
T 04 if





Example 2:

Consider four elements a1, a2, a3 and a4 with Q0 = 1/8, Q1 = 3/16, Q2 = Q3 = Q4 = 1/16 and p1 = 1/4, p2 = 1/8, p3 = p4 =1/16. Construct an optimal binary search tree. Solving for C (0, n):

First, computing all C (i, j) such that j - i = 1; j = i + 1 and as 0 ≤ i < 4; i = 0, 1, 2 and 3; i
< k ≤ J. Start with i = 0; so j = 1; as i < k ≤ j, so the possible value for k = 1

W(0,1)=P(1)+Q(1)+W(0,0)=4+3+2=9

C (0, 1) = W (0, 1) + min {C (0, 0) + C (1, 1)} = 9 + [(0 + 0)] = 9 ft (0, 1) = 1 (value of 'K' that is minimum in the above equation).

Next with i = 1; so j = 2; as i < k ≤ j, so the possible value for k = 2

W(1,2)=P(2)+Q(2)+W(1,1)=2+1+3=6
C (1, 2) = W (1, 2) + min {C (1, 1) + C (2, 2)} = 6 + [(0 + 0)] = 6 ft (1, 2)=2

Next with i = 2; so j = 3; as i < k ≤ j, so the possible value for k = 3

W (2,	3)	=P(3)+Q(3)+W(2,	2)	=1+ 1	+1=3 3)}=3+[(0+0)]=3
C (2 ,	3)	= W (2, 3) + min {C (2,	2)	+C(3,	

ft (2, 3) = 3

Next with i = 3; so j = 4; as i < k ≤ j, so the possible value for k = 4

W (3, 4)	=P(4)+Q(4)+W(3,3)	=1+1	+1=3	
C (3, 4)	= W (3, 4) + min {[C (3, 3)	+C(4,	4)]} =3	+ [(0 + 0)] =3
ft (3, 4)	= 4			

Second, Computing all C (i, j) such that j - i = 2; j = i + 2 and as 0 ≤ i < 3; i = 0, 1, 2; i < k ≤ J

Start with i = 0; so j = 2; as i < k ≤ j, so the possible values for k = 1 and 2.

W(0,2)=P(2)+Q(2)+W(0,1)=2+1+9=12

C (0, 2) = W (0, 2) + min {(C (0, 0) + C (1, 2)), (C (0, 1) + C (2, 2))} = 12 + min {(0 + 6, 9 + 0)} = 12 + 6 = 18

ft (0, 2) = 1

Next, with i = 1; so j = 3; as i < k ≤ j, so the possible value for k = 2 and 3.

W (1, 3) =P(3) +Q(3)+W(1,2)=1+1+6=8

C (1, 3) =W(1, 3) + min {[C (1, 1) + C (2, 3)], [C (1, 2) + C (3, 3)]}

=W(1, 3) + min {(0 + 3), (6 + 0)} = 8 + 3 = 11

ft (1, 3) = 2

Next, with i = 2; so j = 4; as i < k ≤ j, so the possible value for k = 3 and 4.

W(2,4)=P(4)+Q(4)+W(2,3)=1+1+3=5

C (2, 4) = W (2, 4) + min {[C (2, 2) + C (3, 4)], [C (2, 3) + C (4, 4)]

= 5 + min {(0 + 3), (3 + 0)} = 5 + 3 = 8 ft (2, 4) = 3

Third, Computing all C (i, j) such that J - i = 3; j = i + 3 and as 0 ≤ i < 2; i = 0, 1; i < k ≤ J. Start with i = 0; so j = 3; as i < k ≤ j, so the possible values for k = 1, 2 and 3.

W(0,3)=P(3)+Q(3)+W(0,2)=1+1+12=14

C (0, 3) = W (0, 3) + min {[C (0, 0) + C (1, 3)], [C (0, 1) + C (2, 3)], [C (0, 2) + C (3, 3)]}

= 14 + min {(0 + 11), (9 + 3), (18 + 0)} = 14 + 11 = 25 ft (0, 3)=1

Start with i = 1; so j = 4; as i < k ≤ j, so the possible values for k = 2, 3 and 4.

W (1, 4) =P(4)+Q(4)+W(1,3)=1+1+8=10=W

C (1, 4) (1, 4) + min {[C (1, 1) + C (2, 4)], [C (1, 2) +C(3, 4)], [C (1, 3) + C (4, 4)]}

= 10 + min {(0 + 8), (6 + 3), (11 + 0)} = 10 + 8 = 18

ft (1, 4) = 2

Fourth, Computing all C (i, j) such that J - i = 4; j = i + 4 and as 0 ≤ i < 1; i = 0; i < k ≤ J. Start with i = 0; so j = 4; as i < k ≤ j, so the possible values for k = 1, 2, 3 and 4.

W (0, 4) =P(4) +Q(4)+W(0, 3) =1+1 +14=16

C (0, 4) =W(0, 4) + min {[C (0, 0) +C(1, 4)], [C (0, 1) +C(2, 4)], [C (0, 2) +C(3, 4)], [C (0, 3) +C(4, 4)]}

= 16 + min [0 + 18, 9 + 8, 18 + 3, 25 + 0] = 16 + 17 = 33 R (0, 4) = 2

Table for recording W (i, j), C (i, j) and R (i, j)

Column Row	**0**	**1**	**2**	**3**	**4**
0	2,0,0	1,0, 0	1, 0, 0	1, 0, 0,	1,0, 0
1	9,9,1	6,6, 2	3, 3, 3	3, 3, 4	
2	12, 18, 1	8, 11, 2	5, 8, 3		
3	14, 25, 2	11, 18, 2			
4	16, 33, 2				

From the table we see that C (0, 4) = 33 is the minimum cost of a binary search tree for (a1, a2, a3, a4)

The root of the tree 'T04' is 'a2'.

Hence the left sub tree is 'T01' and right sub tree is T24. The root of 'T01' is 'a1' and the root of 'T24' is a3.

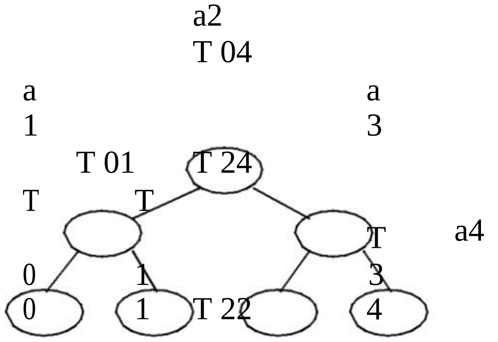
The left and right sub trees for 'T01' are 'T00' and 'T11' respectively. The root of T01 is 'a1'

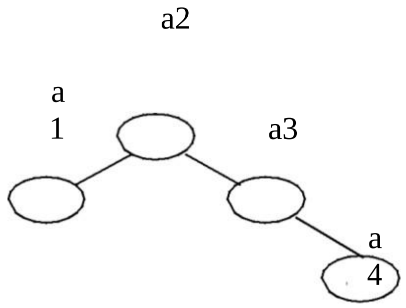
The left and right sub trees for T24 are T22 and T34 respectively.

The root of T24 is 'a3'.

The root of T22 is null.

The root of T34 is a4.





0/1 – KNAPSACK:

We are given n objects and a knapsack. Each object i has a positive weight wi and a positive value Vi. The knapsack can carry a weight not exceeding W. Fill the knapsack so that the value of objects in the knapsack is optimized.

A solution to the knapsack problem can be obtained by making a sequence of decisions on the variables x1, x2, ... , xn. A decision on variable xi involves determining which of the values 0 or 1 is to be assigned to it. Let us assume that decisions on the xi are made in the order xn, xn-1, x1. Following a decision on xn, we may be in one of two possible states: the capacity remaining in m – wn and a profit of pn has accrued. It is clear that the remaining decisions xn-1, , x1 must be optimal with respect to the problem state resulting from the decision on xn. Otherwise, xn,. , x1 will not be optimal. Hence, the principal of optimality holds.

Fn (m) = max {fn-1 (m), fn-1 (m - wn) + pn} -- 1

For arbitrary fi (y), i > 0, this equation generalizes to:

Fi (y) = max {fi-1 (y), fi-1 (y - wi) + pi} -- 2

Equation-2 can be solved for fn (m) by beginning with the knowledge fo (y) = 0 for all y and fi (y) = - ~, y < 0. Then f1, f2, ...fn can be successively computed using equation–2.

When the wi's are integer, we need to compute fi (y) for integer y, 0 $\leq$ y $\leq$ m. Since fi (y)
= - ~ for y < 0, these function values need not be computed explicitly. Since each fi can be computed from fi - 1 in **Θ** (m) time, it takes **Θ** (m n) time to compute fn. When the wi's are real numbers, fi (y) is needed for real numbers y such that 0 < y $\leq$ m. So, fi cannot be explicitly computed for all y in this range. Even when the wi's are integer, the explicit **Θ** (m n) computation of fn may not be the most efficient computation. So, we explore **an alternative method for both cases.**

The fi (y) is an ascending step function; i.e., there are a finite number of y's, 0 = y1 < y2<. . . . < yk, such that fi (y1) < fi (y2) < < fi (yk); fi (y) = - ~ , y < y1; fi (y) = f (yk), y $\geq$ yk; and fi (y) = fi (yj), yj $\leq$ y $\leq$ yj+1. So, we need to compute only fi (yj), 1 $\leq$ j < k. We use the ordered set S^i = {(f (yj), yj) | 1 $\leq$ j $\leq$ k} to represent fi (y). Each number of S^i is a pair (P, W), where P = fi (yj) and W = yj. Notice that S^0 = {(0, 0)}. We can compute S^{i+1} from Si by first computing:

Si 1 = {(P, W) | (P – pi, W – wi) e S^i}

Now, S^{i+1} can be computed by merging the pairs in S^i and Si 1 together. Note that if Si+1 contains two pairs (Pj, Wj) and (Pk, Wk) with the property that Pj $\leq$ Pk and Wj > Wk, then the pair (Pj, Wj) can be discarded because of equation-2. Discarding or purging rules such as this one are also known as dominance rules. Dominated tuples get purged. In the above, (Pk, Wk) dominates (Pj, Wj).

RELIABILITY DESIGN

The problem is to design a system that is composed of several devices connected in series. Let ri be the reliability of device Di (that is ri is the probability that device i will function properly) then the reliability of the entire system is fT ri. Even if the individual devices are very reliable (the ri's are very close to one), the reliability of the system may not be very good. For example, if n = 10 and ri = 0.99, i ≤ i ≤ 10, then fT ri = .904. Hence, it is desirable to duplicate devices. Multiply copies of the same device type are connected in parallel.

If stage i contains mi copies of device Di. Then the probability that all mi have a malfunction is (1 - ri) mi. Hence the reliability of stage i becomes $1 – (1 - r_i)^{mi}$.

The reliability of stage 'i' is given by a function ~i (mi).

Our problem is to use device duplication. This maximization is to be carried out under a cost constraint. Let ci be the cost of each unit of device i and let c be the maximum allowable cost of the system being designed.

We wish to solve:

Max i m iz e ~ *qi* (*mi* ~
1 < *i* < *n*
Subject to ~ *Ci mi* < *C*
1 < *i* < *n*

mi ≥ 1 and interger, 1 ≤ i ≤ n

Assume each Ci > 0, each mi must be in the range 1 ≤ mi ≤ ui, where

~ ~ *n* ~
ui ~ ~ +
~*C* *Ci* ~ *C* ~ ~
C
i
~
J
IL k ~ ~ /U
1 ~

The upper bound ui follows from the observation that mj ≥ 1

An optimal solution m1, m2 . . . . . mn is the result of a sequence of decisions, one decision for each mi.
Let fi (x) represent the maximum value of

Subject to the constrains:

~ *q$* (*mJ*)
1 < *j* < *i*

$\sim C_J \; m_J \sim x$ and $1 \leq m_j \leq u_J$, $1 \leq j \leq i$

$1 < j < i$

The last decision made requires one to choose mn from {1, 2, 3,...........un}

Once a value of mn has been chosen, the remaining decisions must be such as to use the remaining funds C – Cn mn in an optimal way.

The principle of optimality holds on

$f_n \sim C \sim \sim \max \{ On\,(m_n)\, fn\_1\,(C - C_n$
$m_n) \}\; 1 < m_n < u_n$

for any fi (xi), i > 1, this equation generalizes to

$f_n(x) = \max \{ c_i(m_i)\, f_{i-1}(x - C_i$
$m_i) \}\; 1 < m_i < u_i$

clearly, f0 (x) = 1 for all x, 0 ≤ x ≤ C and f (x) = -oo for all x < 0. Let S^i consist of tuples of the form (f, x), where f = fi (x).

There is atmost one tuple for each different ‘x’, that result from a sequence of decisions on m1, m2, ...mn. The dominance rule (f1, x1) dominate (f2, x2) if f1 ≥ f2 and x1 ≤ x2. Hence, dominated tuples can be discarded from S^i.

Example 1:

Design a three stage system with device types D1, D2 and D3. The costs are $30, $15 and $20 respectively. The Cost of the system is to be no more than $105. The reliability of each device is 0.9, 0.8 and 0.5 respectively.

Solution:

We assume that if if stage I has mi devices of type i in parallel, then *0* i (mi) =1 – $(1-ri)^{mi}$

Since, we can assume each ci > 0, each mi must be in the range 1 ≤ mi ≤ ui. Where:

~ ~ *n* - ~ ~
ui = ~ I*C* + *C* ~ *Ci*
Ci ~
~ ~
IL k ~ *J* ~
1 ~

Using the above equation compute u1, u2 and u3.

$$u_1 = \frac{105+30-(30+15+20)}{30} = \frac{70}{30} = 2$$

$$u_2 = \frac{105+15-(30+15+20)}{15} = \frac{55}{15} = 3$$

$$u_3 = \frac{105+20-(30+15+20)}{20} = \frac{60}{20} = 3$$

We useS -* *i*:*stage number and J*: *no. of devices in stage i* = mi *S*°

= {f_o (*x*), *x*}*initially* f_o (*x*) = 1 *and x* = 0, *so*, S^o = {1, 0}

Compute S^1, S^2 and S^3 as follows:

S1 = depends on u1 value, as u1 = 2, so

$S1 = \{S1, S^1\}$

1 2

S2 = depends on u2 value, as u2 = 3, so

$s2=\{S^2,S^2,S^2\}$

1 2 3

S3 = depends on u3 value, as u3 = 3, so

$S3=\{S^3,S^3,S^3\}$

1 2 3

Now find $, 1$ (x), x

S

$f1\ (x) = \{01\ (1)\ f_o \sim \sim,\ 01\ (2)\ f\ 0\ ()\}$ With devices m1 = 1 and m2 = 2 Compute Ø1 (1) and Ø1 (2) using the formula: $Øi\ (mi)) = 1 - (1 - ri\ )\ mi$

~~1~ ~ 1~~1 ~ r ~m 1 = 1 – (1 – 0.9)^1 = 0.9

1

1 1 ~(2) = 1-(1- 0.9) 2

=0.99

S ~ ~$_1f1$ ~x~, x ~ ~

1 ~ ~0.9, 30

S

2

1 =10.99 , 30 + 30 } =(0.99, 60

Therefore, S^1 = {(0.9, 30), (0.99, 60)}

Next find 2 ~ ~~ f

S 1 2 (x), x ~~

$f2\ (x) = \{02\ (1) * f1\ (\ ),\ 02\ (2) * f1\ (\ ),\ 02\ (3) * f1\ (\ )\}$

~2 ~1~ ~ 1 ~ ~1 ~ rI ~ = 1 – (1 – 0.8) = 1 – 0.2 = 0.8

mi 1

~~2~ ~ 1 ~~1 ~ 0.8~ 2 = 0.96 2

$0_2(3)$ = 1 - (1 - 0.8) 3 = 0.992

= {(0.8(0.9),30 + 15), (0.8(0.99),60 + 15)} = {(0.72, 45), (0.792, 75)} = {(0.96(0.9),30 + 15 +15) , (0.96(0.99),60 + 15 + 15)}

= {(0.864, 60), (0.9504, 90)}

= {(0.992(0.9),30 + 15 +15+15) , (0.992(0.99),60 + 15 + 15+15)}

= {(0.8928, 75), (0.98208, 105)}

$S2=\{S^2,S^2,S^2\}$

1 2 3

By applying Dominance rule to S^2:

Therefore, S2 = {(0.72, 45), (0.864, 60), (0.8928, 75)} Dominance Rule:

If S^i contains two pairs (f1, x1) and (f2, x2) with the property that f1 ≥ f2 and x1 ≤ x2, then (f1, x1) dominates (f2, x2), hence by dominance rule (f2, x2) can be discarded. Discarding or pruning rules such as the one above is known as dominance rule. Dominating tuples will be present in S^i and Dominated tuples has to be discarded from Si.

Case 1: if f1 ≤ f2 and x1 > x2 then discard (f1, x1)

Case 2: if f1 ≥ f2 and x1 < x2 the discard (f2, x2)

Case 3: otherwise simply write (f1, x1)

S2 = {(0.72, 45), (0.864, 60), (0.8928, 75)}

Ø 3 (1) = 1 ~ ~1 _ *rI* ~ *mi* = 1 – (1 – 0.5)^1 = 1 – 0.5 = 0.5
Ø 2 ~2~~1~~1=0.75
~*S*0.5~2
3
= 0.875
2
Ø 2 ~3~~1~~1
~ 0.5~ 3
3
3
2

S 13 = {(0.5 (0.72), 45 + 20), (0.5 (0.864), 60 + 20), (0.5 (0.8928), 75 + 20)}

S 13 = {(0.36, 65), (0.437, 80), (0.4464, 95)}

S_2^3 ={(0.75 (0.72), 45 + 20 + 20), (0.75 (0.864), 60 + 20 +20),
(0.75 (0.8928), 75 + 20 + 20)}

= {(0.54, 85), (0.648, 100), (0.6696, 115)}

S 0.875 (0.72), 45 + 20 + 20 + 20), 0.875 (0.864),60 + 20 + 20 + 20),
0.875 (0.8928), 75 + 20 + 20 + 20 }

S 3
3 3 = {

3 = {(0.63, 105), (1.756, 120), (0.7812, 135)} If cost exceeds 105, remove that tuples

S3 = {(0.36, 65), (0.437, 80), (0.54, 85), (0.648, 100)}

The best design has a reliability of 0.648 and a cost of 100. Tracing back for the solution through S^i 's we can determine that m3 = 2, m2 = 2 and m1 = 1.

Other Solution:

According to the principle of optimality:

fn(C) = max {~n (mn). fn-1 (C - Cn mn) with fo (x) = 1 and 0 ≤ x ≤ C; 1 ~ *mn* < *un*

Since, we can assume each ci > 0, each mi must be in the range 1 ≤ mi ≤ ui. Where:
~ (*n* ~ ~

S2 ={(0.75 (0.72), 45 + 20 + 20), (0.75 (0.864), 60 + *u*

= ~ i*C* + *Ci* _ ~*CJ* r / *Ci* I ~
~ *i* ~ ~~
~ ~

Using the above equation compute u1, u2 and u3.

3
105 0 +
u 7
1 0
= 3 = 2
30 ~ 0
u 105 3 5
2 15 0 + 5 ~ 3
= 15 ~
3
0 15 =3
u 105 + 20
3 1 5
= 6
0
=
20 2
0

f3 (105) = max {~3 (m3). f2 (105 - 20m3)} 1 < *m*3 ! *u*3

= max {3(1) f2(105 - 20), 63(2) f2(105 - 20x2), ~3(3) f2(105 -20x3)} = max {0.5

f2(85), 0.75 f2(65), 0.875 f2(45)}

= max {0.5 x 0.8928, 0.75 x 0.864, 0.875 x 0.72} = 0.648.

= max {2 (m2). f1 (85 -15m2)}

1 ! *m*2 ! *u*2

f 2 (8 5)

= max {2(1).f1(85 - 15), ~2(2).f1(85 - 15x2), ~2(3).f1(85 - 15x3)} = max {0.8 f1(70), 0.96 f1(55), 0.992 f1(40)}

= max {0.8 x 0.99, 0.96 x 0.9, 0.99 x 0.9} = 0.8928

f1 (70) = max {~1(m1). f0(70 - 30m1)}

1 ! *m*1 ! *u*1

= max {~1(1) f0(70 - 30), t1(2) f0 (70 - 30x2)}

= max {~1(1) x 1, t1(2) x 1} = max {0.9, 0.99} = 0.99

f1 (55) = max {t1(m1). f0(55 - 30m1)}

1 ! *m*1 ! *u*1

= max {~1(1) f0(50 - 30), t1(2) f0(50 - 30x2)}

= max {~1(1) x 1, t1(2) x -oo} = max {0.9, -oo} = 0.9

f1 (40) = max {~1(m1). f0 (40 - 30m1)}
1 ! *m*1 ! *u*1

= max {~1(1) f0(40 - 30), t1(2) f0(40 - 30x2)}
= max {~1(1) x 1, t1(2) x -oo} = max{0.9, -oo} = 0.9

f2 (65) = max {2(m2). f1(65 -15m2)}
1 ! *m*2 ! *u*2

= max {2(1) f1(65 - 15), 62(2) f1(65 - 15x2), ~2(3) f1(65 - 15x3)} = max {0.8 f1(50),

0.96 f1(35), 0.992 f1(20)}

= max {0.8 x 0.9, 0.96 x 0.9, -oo} = 0.864

f1 (50) = max {~1(m1). f0(50 - 30m1)}
1 ! *m*1 ! *u*1

= max {~1(1) f0(50 - 30), t1(2) f0(50 - 30x2)}

= max {~1(1) x 1, t1(2) x -oo} = max{0.9, -oo} = 0.9 f1 (35) = max ~1(m1). f0(35

- 30m1)}

1 ! *m*1 ! *u*1

= max {~1(1).f0(35-30), ~1(2).f0(35-30x2)}
= max {~1(1) x 1, t1(2) x -oo} = max{0.9, -oo} = 0.9

f1 (20) = max {~1(m1). f0(20 - 30m1)}
1 ! *m*1 ! *u*1

= max {~1(1) f0(20 - 30), t1(2) f0(20 - 30x2)}

= max {~1(1) x -, ~1(2) x -oo} = max{-oo, -oo} = -oo

f2 (45) = max {2(m2). f1(45 -15m2)}
1 ! *m*2 ! *u*2

= max {2(1) f1(45 - 15), ~2(2) f1(45 - 15x2), ~2(3) f1(45 - 15x3)} = max {0.8 f1(30),

0.96 f1(15), 0.992 f1(0)}

= max {0.8 x 0.9, 0.96 x -, 0.99 x -oo} = 0.72

f1 (30) = max {~1(m1). f0(30 - 30m1)} 1 < *m*1 ~ *u*1

= max {~1(1) f0(30 - 30), t1(2) f0(30 - 30x2)}

= max {~1(1) x 1, t1(2) x -oo} = max{0.9, -oo} = 0.9 Similarly, f1 (15) = -,

f1 (0) = -.

The best design has a reliability = 0.648 and

Cost = 30 x 1 + 15 x 2 + 20 x 2 = 100.

Tracing back for the solution through S^i 's we can determine that: m3 = 2, m2 = 2 and m1 = 1.